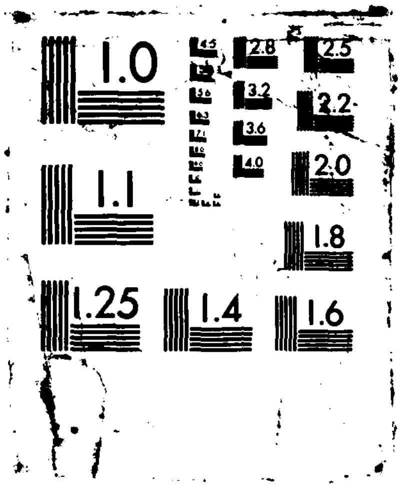
1.0
1.1
1.25
1.4
1.6
1.8
2.0
2.2
2.5
2.8
3.2
3.6
4.0

4

LABORATORY FOR COMPUTER SCIENCE

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

AD-A192 725

MIT/LCS/TM-345

BISIMULATION CAN'T BE TRACED: PRELIMINARY REPORT

Bard Bloom

Sorin Istrail

Albert R. Meyer

November 1987

88 3 1 169

545 TECHNOLOGY SQUARE, CAMBRIDGE, MASSACHUSETTS 02139

Unclassified
SECURITY CLASSIFICATION OF THIS PAGE

REPORT DOCUMENTATION PAGE

1a. REPORT SECURITY CLASSIFICATION Unclassified	1b RESTRICTIVE MARKINGS
2a. SECURITY CLASSIFICATION AUTHORITY	3. DISTRIBUTION/AVAILABILITY OF REPORT Approved for public release; distribution is unlimited.
2b. DECLASSIFICATION/DOWNGRADING SCHEDULE	
4. PERFORMING ORGANIZATION REPORT NUMBER(S) MIT/LCS/TM-345	5. MONITORING ORGANIZATION REPORT NUMBER(S) DARPA/DOD N00014-83-K-0125
6a. NAME OF PERFORMING ORGANIZATION MIT Lab for Computer Science 6b. OFFICE SYMBOL *(If applicable)*	7a. NAME OF MONITORING ORGANIZATION Office of Naval Research/Dept. of Navy
6c. ADDRESS *(City, State, and ZIP Code)* 545 Technology Square Cambridge, MA 02139	7b. ADDRESS *(City, State, and ZIP Code)* Information Systems Program Arlington, VA 22217
8a. NAME OF FUNDING/SPONSORING ORGANIZATION DARPA/DOD 8b. OFFICE SYMBOL *(If applicable)*	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER
8c. ADDRESS *(City, State, and ZIP Code)* 1400 Wilson Blvd. Arlington, VA 22217	10. SOURCE OF FUNDING NUMBERS PROGRAM ELEMENT NO. / PROJECT NO. / TASK NO / WORK UNIT ACCESSION NO.

11. TITLE *(Include Security Classification)*
Bisimulation Can't be Traced: Preliminary Report

12. PERSONAL AUTHOR(S)
Bloom, Bard, Istrail, Sorin, and Meyer, Albert R.

13a. TYPE OF REPORT Technical	13b. TIME COVERED FROM ______ TO ______	14. DATE OF REPORT *(Year, Month, Day)* 1987 November	15 PAGE COUNT 21

16. SUPPLEMENTARY NOTATION

17. COSATI CODES

FIELD	GROUP	SUB-GROUP

18 SUBJECT TERMS *(Continue on reverse if necessary and identify by block number)*
Languages, Semantics, Concurrency, Parallelism, ~~CCS, SCCS~~, ~~CSP, MIEJE~~, bisimulation, observational equivalence, trace equivalence.

19 ABSTRACT *(Continue on reverse if necessary and identify by block number)*
Bisimulation is the primitive notion of equivalence between concurrent processes in Milner's Calculus of Communicating Systems (CCS); there is a nontrivial game-like protocol for distinguishing nonbisimular processes. In contrast, process distinguishability in Hoare's theory of Communicating Sequential Processes (CSP) is determined soley on the basis of traces of visible actions. We examine what additional operations are needed to explain bisimulation similarly-specifically in the case of finitely branching processes without silent moves. We formulate a general notion of Structured Operational Semantics for processes with Guarded recursion (GSOS), and demonstrate that bisimulation does not agree with trace congruence with respect to any set of GSOS-definable contexts. In justifying the generality and significance of GSOS's, we work out some of the basic proof theoretic facts which justify the SOS discipline. Keywords:

20 DISTRIBUTION/AVAILABILITY OF ABSTRACT ☒ UNCLASSIFIED/UNLIMITED ☐ SAME AS RPT. ☐ DTIC USERS	21. ABSTRACT SECURITY CLASSIFICATION Unclassified
22a NAME OF RESPONSIBLE INDIVIDUAL Judy Little	22b TELEPHONE *(Include Area Code)* (617) 253-5894 22c. OFFICE SYMBOL

DD FORM 1473, 84 MAR — 83 APR edition may be used until exhausted. All other editions are obsolete

SECURITY CLASSIFICATION OF THIS PAGE
Unclassified

*U.S. Government Printing Office: 1985-507-047

Bisimulation Can't Be Traced:

Preliminary Report [1]

Bard Bloom [2] — *MIT Lab. for Computer Sci.*
Sorin Istrail — *Wesleyan Univ.*
Albert R. Meyer[3] — *MIT Lab. for Computer Sci.*

Abstract. Bisimulation is the primitive notion of equivalence between concurrent processes in Milner's Calculus of Communicating Systems (CCS); there is a nontrivial game-like protocol for distinguishing nonbisimular processes. In contrast, process distinguishability in Hoare's theory of Communicating Sequential Processes (CSP) is determined solely on the basis of traces of visible actions. We examine what additional operations are needed to explain bisimulation similarly—specifically in the case of finitely branching processes without silent moves. We formulate a general notion of Structured Operational Semantics for processes with Guarded recursion (GSOS), and demonstrate that bisimulation does *not* agree with trace congruence with respect to any set of GSOS-definable contexts. In justifying the generality and significance of GSOS's, we work out some of the basic proof theoretic facts which justify the SOS discipline.

CR Categories and Subject Descriptors: D.3.1 [**Programming Languages**]: Formal Definitions and Theory—*syntax, semantics*; F.3.1 [**Logics and Meanings of Programs**]: Specifying and Verifying and Reasoning about Programs—*program equivalence*; F.3.3 [**Logics and Meanings of Programs**]: Studies of Program Constructs.

General Terms: Languages, Semantics, Concurrency, Parallelism.

Additional Key Words and Phrases: CCS, SCCS, CSP, MIEJE, bisimulation, observational equivalence, trace equivalence.

[1]This Technical Memorandum is a slightly revised version of a paper that will appear under the same title in the Proceedings of the 15th Annual ACM Symposium on Principles of Programming Languages, January, 1988.

[2]Supported by an NSF Fellowship; also NSF Grant No. 8511190-DCR and ONR grant No. N00014-83-K-0125.

[3]Supported by NSF Grand No. 8511190-DCR and by ONR grant No. N00014-83-K-0125.

1 Introduction

Milner's CCS [18], [19] and Hoare's CSP [16], [17] share the premise that the meaning of a process is fully determined by a *synchronization tree*, namely, a rooted, unordered tree whose edges are labeled with symbols denoting basic *actions* or events. These trees are typically specified by a Structured Operational Semantics (SOS) in the style of [23] or by some other effective description, and so are in fact recursively enumerable trees. Both theories further agree that synchronization trees are an *over*specification of process behavior, and certain distinct trees must be regarded as equivalent processes. The notable difference in the theories is that bisimulation yields finer distinctions among synchronization trees.

In CSP, process distinctions can be understood as based on observing *traces*, namely, *maximal* sequences of visible actions performed by a process. Two trees are *trace equivalent* iff they have the same set of traces. Given any set of operations on trees, *trace congruence* is defined to be the coarsest congruence with respect to the operations which refines trace equivalence. Thus, two CSP processes are distinguished iff each can be used in a single CSP context which yields a different set of traces depending on which of the two processes is used. This explanation of when two synchronization trees are to be identified is thoroughly elaborated in Hennessy and DeNicola's *test equivalence* system [10]. On the other hand, two CCS processes are distinguished according to an "interactive" game-like protocol called *bisimulation*. Indistinguishable CCS processes are said to be *bisimular*.

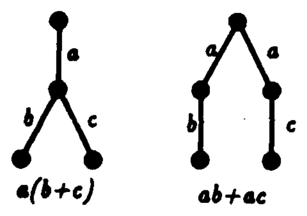

Figure 1: *Trace equivalent but not trace congruent.*

A standard example is the pair of trees (Figure 1) $a(b+c)$ and $(ab+ac)$ which are trace equivalent, but not CSP trace congruent, *viz.*, in CSP (and also CCS) they are distinct processes. Similarly, the trees of (Figure 2)

$$(abc + abd) \text{ and } a(bc + bd) \tag{1}$$

are CSP trace congruent but not bisimular, *viz.*, equal in CSP but distinct in CCS [24], [6]. The trace-based approach is developed in [7], [17], [21], [10]. Bisimulation-based systems include [19], [20], [2], [4], [5], [3].

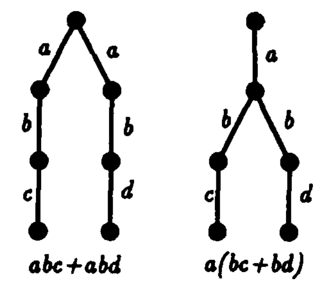

Figure 2: *CSP trace congruent but not bisimular.*

The idea of an "silent" (aka "hidden" or "τ-") action plays an important role in both CSP and CCS theories, but creates significant technical problems. In this paper we assume for simplicity that *there is no silent action*. We expect that our conclusions will generally apply when silent actions can occur, but this remains to be verified.

In the absence of silent action, bisimulation is known to be a congruence with respect to all the operations of CSP/CCS, and Milner has argued extensively that in this case bisimulation yields the finest appropriate notion of the behavior of concurrent processes based on synchronization trees. Although there is some ground for refining synchronization trees further (*cf.* [8]), we shall accept the thesis that bisimular trees are not to be distinguished. Thus, we admit below only operations with respect to which bisimulation remains a congruence. Since bisimular trees are easily seen to be trace equivalent, it follows in this setting that bisimulation refines any trace congruence. Our results focus on the converse question of whether further identifications should be made, *i.e.*, whether nonbisimular processes are truly distinguishable in their observable behavior.

We noted that a pair of nonbisimular trees T_1, T_2 can be distinguished by an "interactive" protocol. The protocol *itself* can be thought of a new process $P[T_1, T_2]$. One might suppose that in a general concurrent programming language, it would be possible to define the new process too, and that success or failure of P running on a pair T_1, T_2 would be easily visible to an observer who could observe traces.

However, CSP and CCS operations are very similar, and the example of Figure 2 above shows that bisimulation is a strictly finer equivalence than trace congruence with respect to CSP/CCS operations. It follows that the contexts P distinguishing nonbisimular processes by their traces *are not definable using the standard CSP/CCS operations*; if they were, nonbisimularity could be reduced to trace distinguishability. Namely, any pair of nonbisimular trees T_1, T_2 would also be trace distinguishable by plugging them in for X in $P[X, T_1]$ and observing the "success" trace when T_1 is plugged in, but not when T_2 is plugged in.

Thus, we maintain that implicit in concurrent process theory based on bisimulation is an-

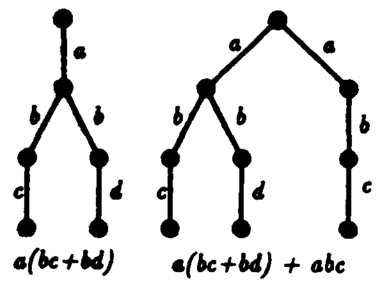

Figure 3: *GSOS congruent but not bisimular.*

other "interactive" kind of *meta*process, which the formalisms of CSP/CCS are inadequate to define! Our question is

> *What further operations on CCS/CSP terms are needed so that protocols reducing nonbisimularity to trace distinguishability become definable?*

In the remainder of the paper, we argue that bisimulation *cannot* be reduced to a trace congruence with respect to any *reasonably structured* system of process constructing operations. The implications of this conclusion are discussed in the final Section 7.

In particular, we formulate in Section 3 a general notion of a system of processes given by structured rules for transitions among terms with guarded recursion—a *GSOS* system. We also indicate why the focus on guarded recursion is both necessary and appropriate. All previously formulated systems of bisimulation-respecting operations are definable by GSOS's. Even rules with negative antecedents are allowed in GSOS's. On the other hand. we indicate in Section 4 that any of the obvious further relaxations of the conditions defining GSOS's can result in systems which are ill-defined or fail to respect bisimulation. Thus, we believe that GSOS definability provides a generous and mathematically invariant constraint on what a reasonably structured system of processes might be.

Definition 1 *Two processes are* GSOS trace congruent *iff they yield the same traces in all GSOS definable contexts.*

Our main result is that bisimulation—even restricted to finite trees—is always a *strict* refinement of GSOS trace congruence. Specifically, we develop in Section 5 a modal logical characterization of GSOS trace congruence similar to the characterization of bisimulation given by Hennessy and Milner [15] and use it to prove:

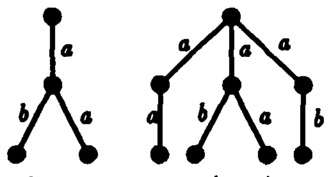

Figure 4: *CSP trace congruent but not GSOS trace congruent.*

Theorem 1 *The nonbisimular trees $a(bc + bd)$ and $a(bc + bd) + abc$ (Figure 3) are GSOS trace congruent.*

We remark that GSOS congruence is a strict refinement of CSP congruence:

Theorem 2 *The processes $aa + ab$ and $aa + ab + a(a + b)$ (see Figure 4) are CSP trace congruent [10, axiom (D5), p. 99] but not GSOS trace congruent.*

Abramsky [1] independently raised the question of how to test distinguishability of nonbisimular processes and formalized the operational behavior of a set of protocols which do capture bisimulation. In Section 6 we offer a similar system for the task, slightly improved in certain respects. Our thesis that no reasonably structured system can capture bisimulation implies that both these systems must lack some important structured features. We also examine the nature of these flaws in detail in Section 6.

2 Trees and Modal Formulas

The simplest tree consists of just a root without edges. It is the "successfully stopped" process, $\mathbf{0}$. If a is an action and P is a tree, then aP is the tree with a fresh root and a single edge, labeled with a, from its root to the root of P (Figure 5). The intention is that aP corresponds to the process which performs action a and then behaves like P. Thus $a\mathbf{0}$ is the tree with one edge labeled a corresponding to the process whose only behavior is to do action a and stop. We ambiguously write just "a" for $a\mathbf{0}$. If P and Q are synchronization trees, let $P + Q$ be the tree obtained by identifying the roots of (disjoint copies of) P and Q (Figure 6). The intention is that $P + Q$ models the nondeterministic process which can behave like either P or Q.

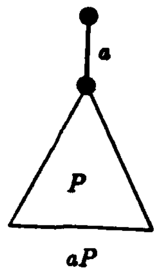

Figure 5: *Tree for aP.*

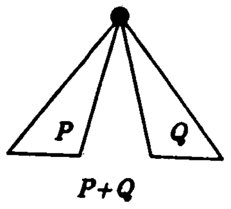

Figure 6: *Tree for $P + Q$.*

This definition of + and **0** explains why both CCS (branching) and CSP (linear) theories accept the familiar axiom

$$p + \mathbf{0} = p\,. \tag{2}$$

Since trees are unordered, both theories also include the equally familiar axioms

$$(p + q) + r = p + (q + r), \quad p + q = q + p\,. \tag{3}$$

Now we note that the tree $a+a$ corresponds intuitively to a process which has two different ways of doing action a and stopping. But the intuition that an action is a minimal detectable event suggests that the theory should disallow detection of the *way* a basic event occurred—all that can be detected is the occurrence of the event itself. Thus in both CSP and CCS, $a = a + a$, and more generally,

$$p = p + p\,, \tag{4}$$

is also accepted as a valid axiom.

The "interactive" protocol defining bisimulation appears in many of the references and we omit it. We remark that two *finite* trees are bisimular iff they are provably equal as a consequence of equations (2)–(4). For our purposes, the most useful formulation of bisimulation is in terms of Hennessy-Milner-logic (HML) formulas [15]:

Definition 2 An *HML formula* is given by the following grammar:

$$\varphi ::= \mathbf{tt} \mid \mathbf{ff} \mid \langle a\rangle\varphi \mid [a]\varphi \mid \varphi\vee\varphi \mid \varphi\wedge\varphi .$$

For any synchronization tree T, the satisfaction relation, $T \models \varphi$, is defined as usual for modal logic and Kripke models (*cf.* [11], [25], [12], [13], or [14]).

A fundamental result of [15] is that two *finitely branching* synchronization trees are bisimular iff they satisfy exactly the same HML formulas.

In fact, both Abramsky's and our operational rules in Section 6 for systems where bisimulation coincides with trace congruence are essentially systems for calculating whether a computably finitely branching synchronization tree satisfies an HML formula.

3 Guarded Terms

Synchronization trees defined in CCS using unguarded recursion may be infinitely branching, corresponding to what are called "unboundedly nondeterministic" processes. Bisimulation between such CCS-definable trees cannot—on purely recursion-theoretic grounds based on degree of undecidability—match trace congruence with respect to any set of effective operations on trees. (Actually, this is an interesting story to work out in detail, but we save that for another paper.)

The high degree of undecidability of bisimulation arises from unbounded nondeterminism. Unbounded nondeterminism is a source of other theoretical difficulties as well, notably that the desired operations on processes are not continuous with respect to any useful known topology. For example, there are simple examples of synchronization trees whose finite subtrees from the root are identical, but which are nevertheless not bisimular. For this reason, restrictions are generally imposed on recursive definitions of processes. In CCS, "guarded" recursion is singled out as attractive, and in CSP and the test-equivalence system of [10], unguarded recursions are treated as though they diverged (with an infinite sequence of silent moves). The essence of these restrictions is to ensure that definable trees behave like *computably finitely branching* trees, *i.e.*, there is an effective procedure which, given (a term denoting) a tree node M, computes the finite set $\left\{N : M \xrightarrow{a} N\right\}$.

For finitely branching trees, trace congruence (including infinite traces) coincides with finite-trace congruence, and as we noted two such trees are bisimular iff they satisfy the same set of HML formulas. Since it is decidable whether a computably finitely branching tree satisfies an HML formula, it follows that bisimulation between such trees is at most Π^0_1, as is the degree of trace congruence with respect to effective operations on such trees. Thus, on recursion-theoretic grounds, finite-trace congruence with respect to some suitable set of operations could equal bisimulation. As noted, this is indeed possible as we show in Section 6. Nevertheless, no reasonably structured operational system can do the job, as we now explain precisely.

We consider theories of synchronization trees described in the standard way by algebraic terms with fixed point operations.

Definition 3 *Process terms*, M, are given by the grammar:

$$M ::= X \mid aM \mid \mathrm{op}(M,\ldots,M) \mid \mathsf{fix}\, X.M \tag{5}$$

where X ranges over synchronization tree variables, a ranges over action symbols, and "op" ranges over operation symbols (of varying arity). M is *guarded* iff, for each subterm $\mathsf{fix}\, X.N$, each occurrence of X in N is within the scope of an a-prefixing operator.

Note that all terms not containing fixed points are guarded. The unary operator $a(\cdot)$ is distinguished from the other operator symbols since it plays a special role in the definition of guarded terms. It will be required to have the same meaning in all GSOS's, namely a-prefixing.

Definition 4 A *Structured Transition Rule* (STR) is a rule of the form:

$$\frac{\left\{X_i \xrightarrow{a_{ij}} Y_{ij} \mid 1 \leq j \leq m_i\right\}_{i=1}^{l}, \quad \left\{X_i \overset{b_{ij}}{\not\rightarrow} \mid 1 \leq j \leq n_i\right\}_{i=1}^{l}}{\mathrm{op}(X_1,\ldots,X_l) \xrightarrow{a} M}$$

where the variables are all distinct, $l \geq 0$ is the arity of op; $m_i, n_i \geq 0$, and M is a guarded term whose free variables are contained in the set of X_i's and Y_{ij}'s. (M need not contain all these variables.) The symbol op is called the *principal operator* of the rule.

The antecedents of the form $X \xrightarrow{a} Y$ are called *positive*, and are satisfied when X and Y are instantiated with processes such that Y is an a-child of X's synchronization tree (*cf.* Definition 7 below). The others, of the form $X \overset{a}{\not\rightarrow}$, are called *negative*; they are satisfied when X is instantiated by a process with no a-children at its root. The rule is *positive* if all the antecedents are positive.

Note that every X_i occurring in the antecedent of an STR must occur as an argument of the principal operator in the consequent, but not every argument of the principal operator need occur in the antecedent.

Definition 5 *Fixed point rules* are of the form:

$$\frac{M[X := \mathsf{fix}\, X.M] \xrightarrow{a} Y}{\mathsf{fix}\, X.M \xrightarrow{a} Y} \tag{6}$$

where $M[X := N]$ denotes substitution of N for X in M, with renaming to avoid capture of free variables in N. *Guarded fixed point rules* are fixed point rules in which the term $\mathsf{fix}\, X.M$ is guarded.

Many of our results are unaffected if an arbitrary family of what we call *δ-rules* of the form

$$\mathbf{c}_i \xrightarrow{a} \mathbf{c}_j$$

are allowed, involving (a possibly uncountable number) of constants $\mathbf{c}_i$.

All GSOS's have all guarded fixed point rules (6), and so these are not mentioned explicitly in the next definition.

Definition 6 A *GSOS rule system* is given by a finite alphabet of actions a, a finite number of STR's, and an arbitrary set of δ-rules. There must, for each a, be a unique rule whose consequent is of the form $aX \xrightarrow{a} M$, and this rule must be

$$aX \xrightarrow{a} X \tag{7}$$

Definition 7 *An* instantiation, *ε, is a map from variables to guarded terms. If P is a guarded term, then $P[\varepsilon]$ denotes the simultaneous substitution of $\varepsilon(X)$ for each free occurrence of X in P.*

Definition 8 *A system of binary relations $\xrightarrow{a}$ on guarded terms, indexed by actions a,* agrees with *a set of rules iff:*

- *Whenever an instantiation by ε of the antecedents of a rule is true of the relation, then the instantiation of the consequent by ε is true as well.*
- *Whenever $P \xrightarrow{a} Q$ is true, then there is a rule r and an instantiation ε such that $P \xrightarrow{a} Q$ is the instantiation of the consequent of r by ε, and the instantiations of the antecedents of r by ε are true.*

Theorem 3 *For any GSOS system, there is a unique system of arrow relations indexed by actions which agrees with the STR's of the GSOS plus the guarded fixed point rules (6).*

Proof: A routine structural induction and slightly less routine fixed point induction on guarded terms. Details omitted. □

For example, Milner's SCCS operators [19] are defined by STR's:

$$\frac{X \xrightarrow{a} Y}{X + X' \xrightarrow{a} Y}, \quad \frac{X' \xrightarrow{a} Y}{X + X' \xrightarrow{a} Y}$$

$$\frac{X \xrightarrow{a} Y}{X{\restriction}\mathsf{A} \xrightarrow{a} Y{\restriction}\mathsf{A}} \; (a \in \mathsf{A} \subseteq \mathsf{Act})$$

$$\frac{X \xrightarrow{a} Y,\ X' \xrightarrow{b} Y'}{X \times X' \xrightarrow{a \cdot b} Y \times Y'}$$

The simple interleaving product of CCS is given by:

$$\frac{X \xrightarrow{a} Y}{X|X' \xrightarrow{a} Y|X'}, \quad \frac{X' \xrightarrow{a} Y}{X|X' \xrightarrow{a} X|Y}$$

Sequential composition of processes, ";", may (and must) be defined using negative rules:

$$\frac{X \xrightarrow{a} Y}{X;X' \xrightarrow{a} Y;X'} \quad \frac{X' \xrightarrow{b} Y',\ \left\{X \not\xrightarrow{a} \;|\; a \in \mathsf{Act}\right\}}{X;X' \xrightarrow{b} Y'}$$

Thus, the operational rules assigning synchronization trees to CCS/CSP/ACP/MIEJE terms easily fit the GSOS framework.

In fact, STR's go beyond the kind of SOS rules needed for CCS in two respects—use of negation and use of *copying*. Namely, there can be more than one antecedent about the behavior of the same subprocess in an STR. For example,

$$\frac{X \xrightarrow{a} Y, X \xrightarrow{b} Y'}{\text{a-if-b}(X) \xrightarrow{a} \text{a-if-b}(Y)}$$

involves two "copies" of process X in the antecedent. Substituting for X in a-if-b($X + b$) trace distinguishes the two trees in Theorem 2 (Figure 4). This example also shows that GSOS congruence strictly refines Phillips *refusal testing* congruence [22] which he was led to develop by considerations similar to ours and Abramsky's. Note that neither recursion nor negative rules were needed in this example. De Simone [9] shows that without negation *or* copying, the only GSOS definable operators are already CCS definable.

4 Why GSOS?

Keeping to the GSOS discipline provides structural induction as a proof technique, as indicated for Theorem 3. In addition, GSOS's guarantee:

Theorem 4 *If a set of δ-rules defines a set of computably finitely branching trees, then the trees definable in any GSOS system with these δ-rules are also computably finitely branching.*

Theorem 5 *Bisimulation is a congruence with respect to all GSOS contexts.*

There are many technical restrictions in our definition of a GSOS rule, and it is natural to ask if they can be relaxed. We indicate how various relaxations may break the key properties of GSOS systems. Note that some systems with non-GSOS rules enjoy the good properties of GSOS systems; however, this is not immediate from the syntactic specifications of these systems. We maintain that GSOS's represent essentially the most general family of systems satisfying Theorems 3-5. The two properties which non-GSOS rules often violate are:

1. The existence of a unique system of arrow relations, $\xrightarrow{a}$, agreeing with the rules.
2. The guarantee that bisimulation is a congruence.

The disjointness of the variables on the right and left sides of arrows in the antecedent is required to guarantee the existence of an arrow relation. Consider a system including the three operators α, β, and γ defined as follows:

$$\frac{X \xrightarrow{a} Y_1, Y_1 \xrightarrow{a} Y_2}{\alpha(X) \xrightarrow{a} \mathbf{0}}$$

$$\frac{X \overset{a}{\nrightarrow}}{\beta(X) \xrightarrow{a} \mathbf{0}}$$

$$\gamma \xrightarrow{a} \beta(\alpha(\gamma))$$

It is not hard to show that there is no arrow relation which agrees with these rules. In particular, $\alpha(\gamma)$ can move iff it cannot move.

Other ways to use variables for pattern-matching allow us to distinguish between bisimular processes. For example, the following rule has two uses of Y.

$$\frac{X_1 \xrightarrow{a} Y, \quad X_2 \xrightarrow{a} Y}{\delta(X_1, X_2) \xrightarrow{a} \mathbf{0}}$$

The context $\delta(a\mathbf{0}, a\cdot)$ distinguishes between the bisimular stopped processes $\mathbf{0}$ and $\mathbf{0}+\mathbf{0}$.

Other forms of pattern-matching, predictably, fail to guarantee preservation of bisimulation. For instance, if we allow the left-hand side of a consequent to look at more than the first operator of a process, we can have the rule

$$\zeta(\mathbf{0}) \xrightarrow{a} \mathbf{0}$$

which again gives a context $\zeta(\cdot)$ which distinguishes between $\mathbf{0}$ and $\mathbf{0}+\mathbf{0}$.

5 Limited Modal Formulas

Definition 9 Limited modal formulas *are given by the following grammar:*

$$\varphi ::= \mathrm{tt} \mid \mathrm{ff} \mid [a]\mathrm{ff} \mid \langle a \rangle \varphi \mid \varphi \wedge \varphi \mid \varphi \vee \varphi.$$

Two trees are defined to be limited modal equivalent *iff they satisfy the same set of limited modal formulas.*

That is, a limited modal formula is an HML formula with a very restricted use of the $[a]$ (necessity) modality.

Theorem 6 *For finitely branching synchronization trees, limited modal equivalence coincides with GSOS trace congruence.*

Proof: A nontrivial structural and fixed-point induction, which is omitted from the preliminary report. □

6 Global Testing Semantics of Bisimulation

Abramsky, "aim[ing] to place the cards on the table as a basis for ... discussion," [1, p. 15], develops a clear system of operational rules whose trace congruence coincides with bisimulation for finitely branching trees without silent actions. By Theorem 1, this system cannot conform to the GSOS discipline. Examining it in detail, we find:

1. There are *two sorts* of processes, those given CCS-like treatment as synchronization trees and other "test" processes which have *different* operations defined on them. For example, test processes are *not* closed under recursive definitions.

2. The inductive rules defining the operational semantics of test processes assign transitions to terms which must pattern-match on more than their outermost operator. This violates the GSOS format and makes it unclear whether the operations respect bisimulation.

3. Indeed, ***bisimulation is not a congruence*** on test processes, nor are bisimular test processes trace congruent.

4. The inductive rules defining the operational semantics of test processes involve *negative antecedents.* This makes it unclear whether the transition relation is even well-defined, given the presence of non-GSOS rules in the system.

5. Finally, to define the operational behavior of tests, Abramsky introduces rule-schemes which he describes as involving *global testing.* Namely, if T is a tree, let $\delta_a(T) = \left\{T_i | T \xrightarrow{a} T_i\right\}$. Then rule schemes of the following form appear:

$$\frac{\delta_a(T) = \{T_1, \ldots, T_n\}}{\exists(T) \xrightarrow{a} \bigvee_{i=1}^{n} \exists(T_i)} \tag{8}$$

   Note that the number of antecedents implicitly appearing in such global-testing rules is *unbounded.*

Items 1, 2, and 3 can be repaired in an *ad hoc* way. A guarded (pun intended) use of negative antecedents can, as already demonstrated in Section 4, be admitted in well-structured systems. We now exhibit our own variation of Abramsky's system as a modest improvement which meets concerns 1–4. The global-testing rules remain, however, as an objectionable feature.

Global testing is objectionable because, in general, ***bisimulation may not be a congruence*** with respect to operations definable with global testing—although it *is* a congruence for our particular system. Indeed it is undecidable to determine whether bisimulation is a congruence given a finite set of GSOS-plus-global-testing rules. Even when bisimulation is a congruence, merely adding positive STRs may destroy the congruence.

Let $\mathcal{S}$ be any GSOS system. We will extend $\mathcal{S}$ to a new system $\bar{\mathcal{S}}$, involving global-testing as well as GSOS rules, in which trace congruence coincides with bisimulation. The extension is conservative, namely, if P is a process in $\mathcal{S}$, then the behavior of P in $\bar{\mathcal{S}}$ is precisely that of P in $\mathcal{S}$.

true	false
and$(\cdot,\cdot)$	or$(\cdot,\cdot)$
pos$_a(\cdot)$	nec$_a(\cdot)$
Sat$(\cdot,\cdot)$	
$\cdot\,\Delta\,\cdot$	$\cdot\,\nabla\,\cdot$

Figure 7: Operators for Global-Testing System

The actions of $\bar{S}$ are the actions of S, plus $\{e, d, t, f, b_1, b_2, \ldots, b_n\}$ where n is a suitably large number chosen below. The operations of $\bar{S}$ are those of S, together with the operators shown in Figure 7

The first six operators are the ***formula-representing operators***; there is a pos$_a(P)$ and nec$_a(P)$ for each action a of $\bar{S}$.

If F codes a HML formula φ, then Sat(P, F) will compute whether or not P satisfies φ.[4] Sat(P, F) takes a t-step if F codes tt, and an f-step if F codes ff; otherwise it makes an e-step, doing one step of the computation trying to see if P satisfies F, and produces an e.

Finally, the binary infix operators $\cdot\,\Delta\,\cdot$ and $\cdot\,\nabla\,\cdot$ are are the ***computational boolean*** operators, which evaluate the conjunctions and disjunctions arising in the evaluation of Sat(P, F).

The system codes HML formulas into bisimulation trees in the following way. Let the ***bit pattern of F***, ***bitpat(F)***, be the vector $\langle v_1, \ldots, v_n\rangle$, where $v_i = 0$ if $F \overset{b_i}{\nrightarrow}$ and $v_i = 1$ otherwise. Choose n large enough so that each connective of HML over $\bar{S}$ can be coded as a distinct bit pattern; let ***code***(σ) be the bit vector coding the connective σ. (The connectives are tt, ff, $\wedge$, $\vee$, and $\langle a\rangle$ and $[a]$ for each action a of $\bar{S}$. Note that the system is slightly self-referential: there are $\langle b_i\rangle$ and $[b_i]$ modalities, coded by the bit patterns of the actions b_i themselves. It is clearly possible to choose n so large that all the necessary modalities may be coded as bit patterns.) To extract a HML formula φ from a process F, we let the bit pattern of F give the leading connective of φ; we write F ***codes*** σ (where σ is a connective) for ***bitpat***(F) = ***code***(σ). Note that F ***codes*** σ may be expressed as a vector of $F \xrightarrow{b_i}$ and $F \overset{b_j}{\nrightarrow}$ conditions, and is meaningful in the antecedent of a GSOS rule.

The arguments, if any, of the main connective of φ will be represented by the children of F under d. For example, the process coding $\varphi_1 \wedge \varphi_2$ is the following. The root can take

[4] F is a metavariable ranging over processes which we intend to be formulas. This is mnemonic only: F may be instantiated by any process. Indeed, this fact forces us to include many of the non-obvious features of the system.

b_i-steps in the bit pattern for $\wedge$. It also has two d-descendants F_1 and F_2, which represent φ_1 and φ_2.

There is an obvious translation $\varphi \mapsto \varphi^*$ of HML formulas into processes. For example,

$$\begin{aligned} (\varphi_1 \wedge \varphi_2)^* &= \text{and}(\varphi_1^*, \varphi_2^*) \\ (\langle a \rangle \varphi)^* &= \text{pos}_a(\varphi^*) \end{aligned}$$

The satisfaction operator Sat(P, F) will attempt to interpret F as a formula of HML, as described above. Sat(P, F) will produce actions e until it decides whether or not P satisfies the formula coded by F; it then produces an action, t or f as appropriate. If F does not code a formula, Sat(P, F) may die without producing a t or f answer, or it may run forever. As it computes, it will make use of the computational Boolean operators $\triangle$ and ∇. For example, we will have the behavior:

$$\text{Sat}(P, (\varphi_1 \wedge \varphi_2)^*) \xrightarrow{e} \text{Sat}(P, \varphi_1^*) \triangle \text{Sat}(P, \varphi_2^*) \tag{9}$$

The computational "and" and "or," $P \triangle Q$ and $P \nabla Q$, combine the e, t, and f actions produced by Sat(P, F). $P \triangle Q$ runs P and Q synchronously until both have finished producing e's; then $P \triangle Q$ produces the conjunction of the truth signals produced by P and Q. $P \nabla Q$ behaves similarly.

The system has some technical properties to facilitate the proof that bisimulation is a congruence. We call a process ***crudely Boolean*** if its synchronization tree consists of a single (finite or infinite) trace, with all actions before the final step being e's, and the final step labeled either e, t, or f. The proofs of congruence depend on having certain processes be crudely Boolean. $P \triangle Q$ and $P \nabla Q$ are crudely Boolean whenever P and Q are; and furthermore, both operators are associative, commutative, and idempotent on crudely Boolean processes. Sat(P, F) is a crude Boolean process for all P and F, in particular for those F which are not the translations of modal formulas.

It is also necessary that an F have a unique sequence of arguments to its coded main connective. $\bar{S}$ follows the convention that the d-descendants of a process are its arguments. However, the S operators in $\bar{S}$ will allow us to build processes which have too many or too few d-descendants; for example,

$$\text{and}(F_0, F_1) + \text{and}(F_2, F_3)$$

codes an ill-formed conjunction with four d-children. We will use global testing to make sure the rules use all the d-descendants as arguments. (Choosing, say, the first two descendants would lead to distinctions between bisimular processes, for bisimular processes may list their descendants in different orders.) The simple rule for conjunctions (equation

9) gives the desired behavior on well-formed conjunctions; in general, we will treat all the d-children of a conjunction as conjuncts.

$$\frac{F\ codes\ \wedge,\quad Children(F,d) = \langle F_1, \ldots, F_n\rangle}{\mathrm{Sat}(P,F) \xrightarrow{e} \Delta_{i=1}^{n}\mathrm{Sat}(P,F_i)}$$

where $\Delta_{i=1}^{n}$ is an iterated Δ, and by convention $\Delta_{i=1}^{0}X_i$ is the process $t\mathbf{0}$. Similarly, we treat all the d-children of a disjunction as disjuncts. A well-formed coded modality $(\langle a\rangle\varphi)^*$ or $([a]\varphi)^*$, has only one d-descendant. Again, to avoid distinguishing bisimular processes representing ill-formed formulas, the rules cannot ignore extra d-descendants; we use the disjunction of all the d-descendants (*cf.* rules (10), (11)), so that $\mathrm{pos}_a F_1 + \mathrm{pos}_a F_2$ is used in much the same way as $\mathrm{pos}_a(\mathrm{or}(F_1, F_2))$. The choice of disjunction was arbitrary; conjunction would have worked as well.

Note also that we are using an ordered global testing rules to make Sat(P,F) have only a single child. We fix some order *e.g.*, lexicographic order on the children of each process when instantiating the antecedent of a global testing rule.

The full set of rules for this system is:

For each connective σ of HML over $\bar{S}$, index i such that the i^{th} bit of $code(\sigma)$ is 1, operator op corresponding to σ, and vector $\vec{X}$ of length equal to the arity of op,

$$\mathrm{op}(\vec{X}) \xrightarrow{b_i} \mathbf{0}$$

In addition,

$$\begin{aligned}
\mathrm{and}(P,Q) &\xrightarrow{d} P\\
\mathrm{and}(P,Q) &\xrightarrow{d} Q\\
\mathrm{or}(P,Q) &\xrightarrow{d} P\\
\mathrm{or}(P,Q) &\xrightarrow{d} Q\\
\mathrm{pos}_a(P) &\xrightarrow{d} P\\
\mathrm{nec}_a(P) &\xrightarrow{d} P
\end{aligned}$$

The rules for $\mathrm{Sat}(P,F)$ are given in Figure 8. Recall that the expression "$P\ codes\ \sigma$" is an abbreviation for a sequence of $P \xrightarrow{b_i} P_i$ and $P \overset{b_i}{\not\rightarrow}$ tests, and therefore is acceptable in the antecedent of a GSOS rule. Also recall that we are treating multiple arguments of a modality as their disjunction.

The computational Boolean operators are largely synchronous, to make them idempotent, associative, and commutative on crudely Boolean processes.

$$\frac{F \textit{ codes } \mathtt{tt}}{\mathrm{Sat}(P,F) \xrightarrow{t} \mathbf{0}}$$

$$\frac{F \textit{ codes } \mathtt{ff}}{\mathrm{Sat}(P,F) \xrightarrow{f} \mathbf{0}}$$

$$\frac{F \textit{ codes } \wedge, \quad \textit{Children}(F,d) = \langle F_1, \ldots, F_n \rangle}{\mathrm{Sat}(P,F) \xrightarrow{e} \Delta_{i=1}^{n} \mathrm{Sat}(P,F_i)}$$

$$\frac{F \textit{ codes } \vee, \quad \textit{Children}(F,d) = \langle F_1, \ldots, F_n \rangle}{\mathrm{Sat}(P,F) \xrightarrow{e} \nabla_{i=1}^{n} \mathrm{Sat}(P,F_i)}$$

$$\frac{F \textit{ codes } \langle a \rangle, \quad \textit{Children}(F,d) = \langle F_1, \ldots, F_n \rangle, \quad \textit{Children}(P,a) = \langle P_1, \ldots, P_m \rangle}{\mathrm{Sat}(P,F) \xrightarrow{e} \nabla_{j=1}^{m} (\nabla_{i=1}^{n} \mathrm{Sat}(P_j, F_i))} \tag{10}$$

$$\frac{F \textit{ codes } [a], \quad \textit{Children}(F,d) = \langle F_1, \ldots, F_n \rangle, \quad \textit{Children}(P,a) = \langle P_1, \ldots, P_m \rangle}{\mathrm{Sat}(P,F) \xrightarrow{e} \Delta_{j=1}^{m} (\nabla_{i=1}^{n} \mathrm{Sat}(P_j, F_i))} \tag{11}$$

Figure 8: *Global-testing rules capturing bisimulation.*

$$\frac{P \xrightarrow{e} P', \quad Q \xrightarrow{e} Q'}{P \nabla Q \xrightarrow{e} P' \nabla Q', \quad P \triangle Q \xrightarrow{e} P' \triangle Q'}$$

$$\frac{P \xrightarrow{e} P', \quad Q \stackrel{e}{\not\rightarrow}}{P \nabla Q \xrightarrow{e} P' \nabla Q, \quad P \triangle Q \xrightarrow{e} P' \triangle Q}$$

$$\frac{P \stackrel{e}{\not\rightarrow}, \quad Q \xrightarrow{e} Q'}{P \nabla Q \xrightarrow{e} P \nabla Q', \quad P \triangle Q \xrightarrow{e} P \triangle Q'}$$

For $a', b' \in \{t, f\}$, let c' be their conjunction and d' their disjunction. We have the rules

$$\frac{P \xrightarrow{a'} P', \quad Q \xrightarrow{b'} Q'}{P \triangle Q \xrightarrow{c'} 0, \quad P \nabla Q \xrightarrow{d'} 0}$$

The following properties of $\bar{S}$ show that it is indeed what we want:

- There is a unique arrow relation compatible with these rules.
- The synchronization tree of each $\bar{S}$-process is computable relative to S, and finitely branching if all S-processes are finitely branching.
- For any formula φ of Hennessy-Milner logic over $\bar{S}$ and process P of $\bar{S}$, the process Sat(P, φ^*) has a trace ending in t iff $P \models \varphi$. So, trace congruence refines bisimulation.
- Bisimulation is a congruence relation with respect to all $\bar{S}$ operators.
- Hence, bisimulation coincides with trace congruence for this system.

The difficult property to verify is that bisimulation is a congruence. A sufficient condition to ensure that a system with global testing respects bisimulation is that the iterated operators in the consequents of global testing rules, in this case $\triangle$ and ∇, are commutative, associative, and idempotent. In our case, commutativity follows routinely from symmetries in the rules. In our system, $\triangle$ and ∇ are also idempotent and associative on crudely Boolean processes, which is all we need for the proof. Our rules were contrived to make this fact easily verified.

1. Trace Equivalence
2. Trace Congruence (CSP)
3. Refusal Testing ([22])
4. GSOS Congruence (Definition [?])
5. Bisimulation (CCS)

Figure 9: Successively Finer Equivalences on Processes

7 Conclusions

Should bisimulation play a significant role in process theory? It has many nice properties, a rich theory, and a tested methodology for verifying correctness of genuine, nontrivial protocols. Nevertheless, we find unconvincing the arguments for taking bisimulation as a primitive notion. We maintain that computational distinctions should be made only because of observable differences "at the terminal". Global testing systems which reduce bisimulation to such observations do not offer what we regard as a reasonable framework for defining operations on processes. Of course, the most persuasive pragmatic argument against bisimulation would be an independently interesting concurrent protocol verifiable using a methodology based on GSOS trace congruence but not even *correct* from the bisimulation view. Such an argument remains to be elaborated.

In any case, the development here clarifies the point of keeping to a GSOS discipline in specifying process behavior. It also motivates further investigation of a new congruence, namely GSOS trace congruence, which is finer than CSP or refusal testing congruence but coarser than bisimulation (cf Figure 9). Particularly noteworthy is the problem of axiomatizing the equational theory of finite trees with the operations $+$ and a-prefixing under GSOS congruence.[5]

References

[1] S. Abramsky. Observation equivalence as a testing equivalence. *Theoretical Computer Sci.*, 00:0–0, 1986. Submitted.

[5] We will present our complete axiomatization of GSOS congruence in a later version of this paper.

[2] D. Austry and G. Boudol. Algèbre de processus et synchronisation. *Theoretical Computer Sci.*, 30:91–131, 1984.

[3] J. C. M. Baeten and R. J. van Glabbeek. Another look at abstraction in process algebra. In T. Ottman, editor, 14th *ICALP*, pages 84–94, Volume 267 of *Lect. Notes in Computer Sci.*, Springer-Verlag, 1987.

[4] J. A. Bergstra and J. W. Klop. Process algebra for synchronous communication. *Information and Control*, 60:109–137, 1984.

[5] J. A. Bergstra and J. W. Klop. Algebra of communicating processes with abstraction. *Theoretical Computer Sci.*, 37:77–121, 1985.

[6] S. D. Brookes. On the relationship of CCS and CSP. In 10th *ICALP*, pages 83–96, Springer-Verlag, 1983.

[7] S. D. Brookes, C. A. R. Hoare, and A. W. Roscoe. A theory of communicating sequential processes. *J. ACM*, 31:560–569, 1984.

[8] L. Castellano, G. D. Michelis, and L. Pomello. Concurrency vs interleaving: an instructive example. *Bull. Europ. Ass. Theoretical Computer Sci.*, (31):12–15, 1987.

[9] R. de Simone. Higher-level synchronising devices in MIEJE-SCCS. *Theoretical Computer Sci.*, 37:245–267, 1985.

[10] R. DeNicola and M. C. Hennessy. Testing equivalences for processes. *Theoretical Computer Sci.*, 34:83–133, 1984.

[11] M. Fischer and R. Ladner. Propositional dynamic logic of regular programs. *J. Computer and System Sci.*, 18:194–211, 1979.

[12] S. Graf and J. Sifakis. From synchronisation tree logic to acceptance. In R. Parikh, editor, *Logics of Programs*, pages 128–142, Volume 193 of *Lect. Notes in Computer Sci.*, Springer-Verlag, 1985.

[13] S. Graf and J. Sifakis. A modal characterization of observational congruence on finite terms of CCS. *Information and Control*, 125–145, 1986.

[14] S. Graf and J. Sifakis. Readiness semantics for regular processes with silent actions. In T. Ottman, editor, 14th *ICALP*, pages 115–125, Volume 267 of *Lect. Notes in Computer Sci.*, Springer-Verlag, 1987.

[15] M. C. B. Hennessy and R. Milner. Algebraic laws for nonderminism and concurrency. *J. ACM*, 32:137–161, 1985.

[16] C. Hoare. Communicating sequential processes. *Comm. ACM*, 21:666–677, 1978.

[17] C. A. R. Hoare. *Communicating Sequential Processes. Series in Computer Science*, Prentice-Hall, 1985. 256 pp.

[18] R. Milner. *A Calculus of Communicating Systems.* Volume 92 of *Lect. Notes in Computer Sci.*, Springer-Verlag, 1980. 171 pp.

[19] R. Milner. Calculi for synchrony and asynchrony. *Theoretical Computer Sci.*, 25:267–310, 1983.

[20] R. Milner. Lectures on a calculus for communicating systems. In S. Brookes, A. Roscoe, and G. Winskel, editors, *Seminar on Concurrency*, pages 197–220, Volume 197 of *Lect. Notes in Computer Sci.*, Springer-Verlag, 1984.

[21] E. Olderog and C. A. R. Hoare. Specification-oriented semantics for communicating processes. *Acta Informatica*, 23:9–66, 1986.

[22] I. Phillips. Refusal testing. In 13^{th} *ICALP*, pages 304–313, Volume 226 of *Lect. Notes in Computer Sci.*, Springer-Verlag, 1986.

[23] G. D. Plotkin. *A structural approach to operational semantics.* Technical Report DAIMI FN-19, Aarhus Univ., Computer Science Dept., Denmark, 1981.

[24] A. Pnueli. Linear and branching structures in the semantics and logics of reactive systems. In W. Brauer, editor, *Int'l Conf. Automata, Languages and Programming*, pages 15–32, Springer-Verlag, 1985.

[25] R. S. Streett. PDL of looping and converse is elementarily decidable. *Information and Control*, 54:121–141, 1982.

Cambridge, Massachusetts, November 3, 1987

OFFICIAL DISTRIBUTION LIST

Director 2 Copies
Information Processing Techniques Office
Defense Advanced Research Projects Agency
1400 Wilson Boulevard
Arlington, VA 22209

Office of Naval Research 2 Copies
800 North Quincy Street
Arlington, VA 22217
Attn: Dr. R. Grafton, Code 433

Director, Code 2627 6 Copies
Naval Research Laboratory
Washington, DC 20375

Defense Technical Information Center 12 Copies
Cameron Station
Alexandria, VA 22314

National Science Foundation 2 Copies
Office of Computing Activities
1800 G. Street, N.W.
Washington, DC 20550
Attn: Program Director

Dr. E.B. Royce, Code 38 1 Copy
Head, Research Department
Naval Weapons Center
China Lake, CA 93555

Dr. G. Hooper, USNR 1 Copy
NAVDAC-OOH
Department of the Navy
Washington, DC 20374